[0120] Peripheral hardware environments, on the other hand, change more frequently. E.g., workload bearing (line/CPU) cards (e.g., blades) are often added/removed; mezzanine card slots/headers can be filled with optional hardware products; disk drives come and go; etc.

[0121] Accordingly, in some embodiments, the present invention supports changing peripheral hardware environments by providing for automating changes to the description block when needed (see **FIG. 3**). In particular, in some embodiments, the present invention supports discovery of changes and related updates. Discovery, in this context, involves detecting changes in the hardware environment. Those skilled in the art will understand that there are many ways to implement discovery. Most discovery technologies are very particular to a specific hardware environment. For instance, some hardware is designed to raise some form of asynchronous event (via some interrupt line, an I2C bus message or some other form of input). (An I2C (Inter-IC) bus is a bi-directional two-wire serial bus that provides a communication link between integrated circuits (ICs).) Some convention exists to then name/identify the hardware changes announced by this asynchronous event.

[0122] Regardless of how discovery takes place, this invention provides for the ability to address discovered changes by simply activating/de-activating different peripheral device descriptions. Alternatively, in some embodiments, this invention can obtain needed peripheral device descriptions from a remote source (e.g., over a network, via a serial port or the like). The Dynamic Discovery Monitor (DMM) according to embodiments of the present invention is shown in **FIG. 3**.

[0123] Description changes themselves can trigger associated logic—which typically employs some event-condition-action (ECA) trigger mechanism/method. In some aspects, embodiments of the present invention include the option of creating a so-called "active" repository of logical/physical descriptions. This amounts to an active description block (see **FIG. 3**). Again, this can be done without any change to the nature/size/layout of fixed software/firmware image. No new software/firmware needs to be dynamically linked or dynamically loaded. There is no need for a software/firmware re-initialization.

[0124] The block-of-bits may be considered to be like an opaque union of all the different data structures that this block of bits (64-256, or more) might represent—within a given schema. The layout/contents of the block-of-bits depends upon the logical/physical description scheme being used. Facet logic (from getters/setters to operational behaviors—like power on/off/recycle) always applies to a given scheme (with the schema). Since the facet logic knows what assumptions it can make about its scheme, at run-time it can appropriately cast the block-of-bits it receives.

[0125] Most logic is higher-level than facet logic, which means that most logic treats the block-of-bits as opaque. Most logic just feeds in the given instance's corresponding description details/values into facets as the block-of-bits. Most logic knows nothing about how to interpret/use this block-of-bits. In this sense, the block-of-bits is usually opaque. However, the specific facet logic does know just how the block-of-bits is laid out (and thus how to interpret/use it).

[0126] Aspects of the present invention may be considered to be a way to declaratively describe entities with a nested/hierarchy of prototypes (rather than types), along with ways to make all prototype slot/facet interfaces uniform/consistent (within a given over-arching schema). Within a given schema, some number of bits (e.g., 64, 128 or 256) are always passed as a universal/singular argument to all prototyped behaviors/methods/functions.

[0127] The binding of a type to the usually-opaque block is delayed, past run-time initialization, all the way down to the entry point of scheme-specific, scheme-facet-implementation module. This type-to-block-of-bits binding is repeated whenever this module is entered. By late binding in this manner, the present invention provides a way to greatly expand the expressive range of declarative (prototyped) schema. This, in turn allows data-driven firmware/software logic, according to embodiments of the present invention, to be far more widely applicable which, in turn, makes a single, fixed-size firmware/software image far more flexible/applicable/valuable.

[0128] Although the present invention has been described with reference to specific exemplary embodiments and examples, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention as set forth in the claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than restrictive sense.

We claim:

1. A method for producing a fixed-size firmware image for a hardware device, parameterized for a plurality of component environments, the method comprising:

providing a logical description of aspects of said plurality of component environments;

providing a physical description of physical aspects of said plurality of component environments;

associating said logical description with said physical description;

providing said firmware image to include a plurality of parameterized functions to support said hardware device in each of said plurality of component environments.

2. A method as in claim 1 wherein said hardware device is selected from the group comprising: power supplies, busses; fans, disk drives, sensors, and flash parts.

3. A method as in claim 1 wherein actual arguments to said parameterized functions are bound at run time.

4. A method as in claim 1 wherein actual argument to each of said parameterized functions is a fixed-sized block of bits whose interpretation is context sensitive.

5. A method as in claim 4 wherein each block of bits is the same fixed size.

6. A method as in claim 5 wherein each block of bits is between 64 and 256 bits long.

7. A method as in claim 1 wherein the firmware image comprises:

an operational block including abstract device driver interfaces for said hardware device; and

a description block that includes said logical and physical descriptions.

8. A method as in claim 4 wherein the actual argument to each of said parameterized functions is cast at run-time into a run-time context determined type.